

In The Name Of Allah

Chapter 1
Number Systems and Codes

Number Systems (1)

➤ *Positional Notation*

$$N = (a_{n-1}a_{n-2} \dots a_1a_0 . a_{-1}a_{-2} \dots a_{-m})_r \quad (1.1)$$

where $.$ = radix point

r = radix or base

n = number of integer digits to the left of the radix point

m = number of fractional digits to the right of the radix point

a_{n-1} = most significant digit (MSD)

a_{-m} = least significant digit (LSD)

➤ *Polynomial Notation* (Series Representation)

$$\begin{aligned} N &= a_{n-1} \times r^{n-1} + a_{n-2} \times r^{n-2} + \dots + a_0 \times r^0 + a_{-1} \times r^{-1} \dots + a_{-m} \times r^{-m} \\ &= \sum_{i=-m}^{n-1} a_i r^i \end{aligned} \quad (1.2)$$

$$N = (251.41)_{10} = 2 \times 10^2 + 5 \times 10^1 + 1 \times 10^0 + 4 \times 10^{-1} + 1 \times 10^{-2}$$

Number Systems (2)

➤ *Binary* numbers

- Digits = {0, 1}

- $(11010.11)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$
 $= (26.75)_{10}$

- 1 K (kilo) = $2^{10} = 1,024$, 1M (mega) = $2^{20} = 1,048,576$,
1G (giga) = $2^{30} = 1,073,741,824$

➤ *Octal* numbers

- Digits = {0, 1, 2, 3, 4, 5, 6, 7}

- $(127.4)_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$

➤ *Hexadecimal* numbers

- Digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

- $(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46,687)_{10}$

Number Systems (3)

➤ *Important Number Systems* (Table 1.1)

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	1	F
16	10000	20	10

Arithmetic (1)

➤ *Binary Arithmetic*

▪ *Addition*

111011	Carries
101011	Augend
+ 11001	Addend
1000100	

▪ *Subtraction*

0	1	10	0	10	Borrows	
1	0	0	1	0	1 Minuend	
-		1	1	0	1	Subtrahend
			1	0	1	0

Arithmetic (3)

➤ *Octal Arithmetic* (Use Table 1.4)

▪ *Addition*

$$\begin{array}{r}
 1 \ 1 \ 1 \qquad \text{Carries} \\
 5 \ 4 \ 7 \ 1 \ \text{Augend} \\
 + \ 3 \ 7 \ 5 \ 4 \ \text{Addend} \\
 \hline
 11445 \ \text{Sum}
 \end{array}$$

▪ *Subtraction*

$$\begin{array}{r}
 6 \ 14 \ 4 \ 11 \ \text{Borrows} \\
 7 \ 4 \ 5 \ 1 \ \text{Minuend} \\
 - \ 5 \ 6 \ 4 \ 3 \ \text{Subtrahend} \\
 \hline
 1 \ 6 \ 0 \ 6 \ \text{Difference}
 \end{array}$$

Arithmetic (4)

■ *Multiplication*

$$\begin{array}{r}
 326 \\
 \times 67 \\
 \hline
 2732 \\
 2404 \\
 \hline
 26772
 \end{array}$$

Multiplicand

Multiplier

Partial products

Product

Division

Dividend	63) 7514	Quotient
		114	
		63	
		114	
		63	
		364	
		314	
		50	Remainder

Arithmetic (5)

➤ *Hexadecimal Arithmetic* (Use Table 1.5)

▪ *Addition*

$$\begin{array}{r}
 1\ 0\ 1\ 1\ \text{Carries} \\
 5\ B\ A\ 9\ \text{Augend} \\
 +\ D\ 0\ 5\ 8\ \text{Addend} \\
 \hline
 1\ 2\ C\ 0\ 1\ \text{Sum}
 \end{array}$$

▪ *Subtraction*

$$\begin{array}{r}
 9\ 15\ A\ 19\ \text{Borrows} \\
 A\ 5\ B\ 9\ \text{Minuend} \\
 +\ 5\ 8\ 0\ D\ \text{Subtrahend} \\
 \hline
 4\ D\ A\ C\ \text{Difference}
 \end{array}$$

Arithmetic (6)

▪ *Multiplication*

B9A5
 x D50

 3A0390
 96D61

 9A76490

Multiplicand

Multiplier

Partial products

Product

Division

Dividend 57F6D
 Divisor B9)

 79B Quotient
 50F

 706
 681

 85D
 7F3

 6A Remainder

Base Conversion (1)

➤ *Series Substitution Method*

- Expanded form of polynomial representation:

$$N = a_{n-1}r^{n-1} + \dots + a_0r^0 + a_{-1}r^{-1} + \dots + a_{-m}r^{-m} \quad (1.3)$$

- Conversion Procedure (base A to base B)

- o Represent the number in base A in the format of Eq. 1.3.
- o Evaluate the series using base B arithmetic.

- Examples:**

- o $(11010)_2 = (?)_{10}$

$$\begin{aligned} N &= 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 \\ &= (16)_{10} + (8)_{10} + 0 + (2)_{10} + 0 \\ &= (26)_{10} \end{aligned}$$

- o $(627)_8 = (?)_{10}$

$$\begin{aligned} N &= 6*8^2 + 2*8^1 + 7*8^0 \\ &= (384)_{10} + (16)_{10} + (7)_{10} \\ &= (407)_{10} \end{aligned}$$

Base Conversion (2)

➤ *Radix Divide Method*

- Used to convert the integer in base A to the equivalent base B integer.

- Underlying theory:

$$\circ (N)_A = b_{n-1}B^{n-1} + \dots + b_0B^0 \quad (1.4)$$

Here, b_i 's represents the digits of $(N)_B$ in base A .

$$\begin{aligned} \circ N_i / B &= (b_{n-1}B^{n-1} + \dots + b_1B^1 + b_0B^0) / B \\ &= (\text{Quotient } Q_i: b_{n-1}B^{n-2} + \dots + b_1B^0) + (\text{Remainder } R_0: b_0) \end{aligned}$$

- In general, $(b)_A$ is the remainder R_i when Q_i is divided by $(B)_A$.

- *Conversion Procedure*

1. Divide $(N)_B$ by $(B)_A$, producing Q_1 and R_0 . R_0 is the least significant digit, d_0 , of the result.
2. Compute d_i for $i = 1 \dots n - 1$, by dividing Q_i by $(B)_A$, producing Q_{i+1} and R_i which represents d_i .
3. Stop when $Q_{i+1} = 0$.

Base Conversion (3)

▪ *Examples*

$$o(315)_{10} = (473)_8$$

8	315	3	↑	LSD
	8	39	7	
	8	4	4	MSD
		0		

$$o(315)_{10} = (13B)_{16}$$

16	315	B	↑	LSD
	16	19	3	
	16	1	1	MSD
		0		

Base Conversion (4)

➤ *Radix Multiply Method*

- Used to convert fractions.

- Underlying theory:

- $(N_F)_A = b_{-1}B^{-1} + b_{-2}B^{-2} + \dots + b_{-m}B^{-m}$ (1.5)

- Here, $(N_F)_A$ is a fraction in base A and b_i 's are the digits of $(N_F)_B$ in base A .

- $B \cdot N_F = B \cdot (b_{-1}B^{-1} + b_{-2}B^{-2} + \dots + b_{-m}B^{-m})$
 $= (\text{Integer } I_{-1}; b_{-1}) + (\text{Fraction } F_{-2}; b_{-2}B^{-1} + \dots + b_{-m}B^{-(m-1)})$

- In general, $(b_i)_A$ is the integer part I_{-i} of the product of $F_{-(i+1)} \cdot (B_A)$.

- *Conversion Procedure*

1. Let $F_{-1} = (N_F)_A$.

2. Compute digits $(b_{-i})_A$, for $i = 1 \dots m$, by multiplying F_i by $(B)_A$, producing integer I_{-i} , which represents $(b_{-i})_A$, and fraction $F_{-(i+1)}$.

3. Convert each digits $(b_{-i})_A$ to base B .

Base Conversion (5)

▪ *Examples*

$$\circ(0.479)_{10} = (0.3651\dots)_8$$

$$\text{MSD} \quad 3.832 = 0.479 * 8$$

$$6.656 = 0.832 * 8$$

$$5.248 = 0.656 * 8$$

$$\text{LSD} \quad 1.984 = 0.248 * 8$$

...

$$\circ(0.479)_{10} = (0.0111\dots)_2$$

$$\text{MSD} \quad 0.9580 = 0.479 * 2$$

$$1.9160 = 0.9580 * 2$$

$$1.8320 = 0.9160 * 2$$

$$\text{LSD} \quad 1.6640 = 0.8320 * 2$$

...

Base Conversion (6)

➤ *General Conversion Algorithm*

➤ *Algorithm 1.1*

To convert a number N from base A to base B , use

- (a) the series substitution method with base B arithmetic, or
- (b) the radix divide or multiply method with base A arithmetic.

➤ *Algorithm 1.2*

To convert a number N from base A to base B , use

- (a) the series substitution method with base 10 arithmetic to convert N from base A to base 10, and
- (b) the radix divide or multiply method with decimal arithmetic to convert N from base 10 to base B .

➤ Algorithm 1.2 is longer, but easier and less error prone.

Base Conversion (7)

➤ *Example*

$$(18.6)_9 = (?)_{11}$$

(a) Convert to base 10 using series substitution method:

$$\begin{aligned} N_{10} &= 1 * 9^1 + 8 * 9^0 + 6 * 9^{-1} \\ &= 9 + 8 + 0.666... \\ &= (17.666...)_{10} \end{aligned}$$

(b) Convert from base 10 to base 11 using radix divide and multiply method:

$$7.326 = 0.666 * 11$$

$$3.586 = 0.326 * 11$$

$$6.446 = 0.586 * 11$$

$$\begin{array}{r} 11 \overline{) 17} \\ \underline{11} \\ 6 \end{array} \quad \begin{array}{r} 6 \\ 1 \end{array} \begin{array}{l} \cdot \\ \downarrow \end{array}$$

$$N_{11} = (16.736 \dots)_{11}$$

Base Conversion (8)

➤ When $B = A^k$

➤ **Algorithm 1.3**

(a) To convert a number N from base A to base B when $B = A^k$ and k is a positive integer, group the digits of N in groups of k digits in both directions from the radix point and then replace each group with the equivalent digit in base B

(b) To convert a number N from base B to base A when $B = A^k$ and k is a positive integer, replace each base B digit in N with the equivalent k digits in base A .

Examples

- $(001\ 010\ 111.\ 100)_2 = (127.4)_8$ (group bits by 3)
- $(1011\ 0110\ 0101\ 1111)_2 = (B65F)_{16}$ (group bits by 4)

Signed Number Representation

➤ *Signed Magnitude Method*

- $N = \pm (a_{n-1} \dots a_0 \cdot a_{-1} \dots a_{-m})_r$ is represented as

$$N = (sa_{n-1} \dots a_0 \cdot a_{-1} \dots a_{-m})_{rsm} \quad (1.6)$$

where $s = 0$ if N is positive and $s = r - 1$ otherwise.

- $N = -(15)_{10}$
- In binary: $N = -(15)_{10} = -(1111)_2 = (1, 1111)_{2sm}$
- In decimal: $N = -(15)_{10} = (9, 15)_{10sm}$

➤ *Complementary Number Systems*

- *Radix complements* (r 's complements)

$$[M]_r = r^n - (M)_r \quad (1.7)$$

where n is the number of digits in $(M)_r$

- *Positive full scale*: $r^{n-1} - 1$
- *Negative full scale*: $-r^n - 1$
- *Diminished radix complements* ($r-1$'s complements)

$$[M]_{r-1} = r^n - (M)_r - 1$$

Radix Complement Number Systems (1)

- Two's complement of $(M)_2 = (101001)_2$
 $[M]_2 = 2^6 - (101001)_2 = (1000000)_2 - (101001)_2 = (010111)_2$
- $(M)_2 + [M]_2 = (101001)_2 + (010111)_2 = (1000000)_2$
 If we discard the carry, $(M)_2 + [M]_2 = 0$.
 Hence, $[M]_2$ can be used to represent $-(M)_2$.
- $[[M]_2]_2 = [(010111)_2]_2 = (1000000)_2 - (010111)_2 = (101001)_2 = (M)_2$.
- Two's complement of $(M)_2 = (1010)_2$ for $n = 6$
 $[M]_2 = (1000000)_2 - (1010)_2 = (110110)_2$.
- Ten's complement of $(M)_{10} = (72092)_{10}$
 $[M]_{10} = (100000)_{10} - (72092)_{10} = (27908)_{10}$.

Radix Complement Number Systems (2)

➤ **Algorithm 1.4** Find $[M]_r$ given $(N)_r$.

- Copy the digits of N , beginning with the LSD and proceeding toward the MSD until the first nonzero digit, a_i , has been reached
- Replace a_i with $r - a_i$.
- Replace each remaining digit a_j , of N by $(r - 1) - a_j$ until the MSD has been replaced.

➤ **Example:** 10's complement of $(56700)_{10}$ is $(43300)_{10}$

➤ **Example:** 2's complement of $(10100)_2$ is $(01100)_2$.

➤ **Example:** 2's complement of $N = (10110)_2$ for $n = 8$.

- Put three zeros in the MSB position and apply algorithm 1.4
- $N = 00010110$
- $[M]_2 = (11101010)_2$

➤ The same rule applies to the case when N contains a radix point.

Radix Complement Number Systems (3)

➤ **Algorithm 1.5** Find $[M]_r$ given $(M)_r$.

- First replace each digit, a_k , of $(M)_r$ by $(r - 1) - a_k$ and then add 1 to the resultant.

➤ For binary numbers ($r = 2$), complement each digit and add 1 to the result.

➤ **Example:** Find 2's complement of $N = (01100101)_2$.

$$N = 01100101$$

$$10011010 \quad \text{Complement the bits}$$

$$+1 \quad \text{Add 1}$$

$$[M]_2 = (10011011)_{10}$$

➤ **Example:** Find 10's complement of $N = (40960)_{10}$

$$N = 40960$$

$$59039 \quad \text{Complement the bits}$$

$$+1 \quad \text{Add 1}$$

$$[M]_{10} = (59040)_{10}$$

Radix Complement Number Systems (4)

➤ *Two's complement number system* (See Table 1.6):

▪ Positive number :

$$N = +(a_{n-2}, \dots, a_0)_2 = (0, a_{n-2}, \dots, a_0)_{2cns}$$

$$0 \leq N \leq 2^{n-1} - 1$$

where

▪ Negative number:

$$N = (a_{n-1}, a_{n-2}, \dots, a_0)_2$$

$$-1 \geq N \geq -2^{n-1}$$

$$-N = [a_{n-1}, a_{n-2}, \dots, a_0]_2 \text{ (two's complement of } N),$$

where

▪ **Example:** Two's complement number system representation of $\pm (M)_2$

when $(M)_2 = (1011001)_2$ for $n = 8$:

$$+(M)_2 = (0, 1011001)_{2cns}$$

$$-(M)_2 = [+(M)_2]_2 = [0, 1011001]_2 = (1, 0100111)_{2cns}$$

Radix Complement Number Systems (5)

➤ **Example:** Two's complement number system representation of $-(18)_{10}$, $n = 8$:

- $+(18)_{10} = (0, 0010010)_{2cns}$

- $-(18)_{10} = [0, 0010010]_2 = (1, 1101110)_{2cns}$

➤ **Example:** Decimal representation of $N = (1, 1101000)_{2cns}$

- $N = (1, 1101000)_{2cns} = -[1, 1101000]_2 = -(0, 0011000)_{2cns} = -(24)_2$.

Radix Complement Arithmetic (1)

- Radix complement number systems are used to convert subtraction to addition, which reduces hardware requirements (only adders are needed).
- $A - B = A + (-B)$ (add r 's complement of B to A)
- Range of numbers in two's complement number system: , where n is the number of bits. $-2^{n-1} \leq N \leq 2^{n-1} - 1$
- $2^{n-1} - 1 = (0, 11 \dots 1)_{2\text{cns}}$ and $-2^{n-1} = (1, 00 \dots 0)_{2\text{cns}}$
- If the result of an operation falls outside the range, an **overflow condition** is said to occur and the result is not valid.
- Consider three cases:
 - $A = B + C,$
 - $A = B - C,$
 - $A = -B - C,$
 (where $B \geq 0$ and $C \geq 0$.)

Radix Complement Arithmetic (2)

➤ Case 1: $A = B + C$

- $(A)_2 = (B)_2 + (C)_2$
- If $A > 2^{n-1} - 1$ (**overflow**), it is detected by the n th bit, which is set to 1 (**in corrected**).

- **Example:** $(7)_{10} + (4)_{10} = ?$ using 5-bit two's complement arithmetic.
 - o $(7)_{10} = +(0111)_2 = (0, 0111)_{2cns}$
 - o $(4)_{10} = +(0100)_2 = (0, 0100)_{2cns}$
 - o $(0, 0111)_{2cns} + (0, 0100)_{2cns} = (0, 1011)_{2cns} = +(1011)_2 = +(11)_{10}$
 - o No overflow.

- **Example:** $(9)_{10} + (8)_{10} = ?$
 - o $(9)_{10} = +(1001)_2 = (0, 1001)_{2cns}$
 - o $(8)_{10} = +(1000)_2 = (0, 1000)_{2cns}$
 - o $(0, 1001)_{2cns} + (0, 1000)_{2cns} = (1, 0001)_{2cns}$ (**overflow**)

Radix Complement Arithmetic (3)

➤ Case 2: $A = B - C$

- $A = (B)_2 + (-(C)_2) = (B)_2 + [C]_2 = (B)_2 + 2^n - (C)_2 = 2^n + (B - C)_2$
- If $B > C$, then $A > 2^n$ and the carry is discarded.
- So, $(A)_2 = (B)_2 + [C]$ | carry discarded
- If $B < C$, then $A = 2^n - (C - B)_2 = [C - B]_2$ or $A = -(C - B)_2$ (no carry in this case).
- No overflow for Case 2.

▪ **Example:** $(14)_{10} - (9)_{10} = ?$

- Perform $(14)_{10} + (-(9)_{10})$
- $(14)_{10} = +(1110)_2 = (0, 1110)_{2cns}$
- $-(9)_{10} = -(1001)_2 = (1, 0111)_{2cns}$
- $(14)_{10} - (9)_{10} = (0, 1110)_{2cns} + (1, 0111)_{2cns} = (0, 0101)_{2cns} + carry$
 $= +(0101)_2 = +(5)_{10}$

Radix Complement Arithmetic (4)

▪ **Example:** $(9)_{10} - (14)_{10} = ?$

o Perform $(9)_{10} + (-(14)_{10})$

o $(9)_{10} = +(1001)_2 = (0, 1001)_{2cns}$

o $-(14)_{10} = -(1110)_2 = (1, 0010)_{2cns}$

o $(9)_{10} - (14)_{10} = (0, 1001)_{2cns} + (1, 0010)_{2cns} = (1, 1011)_{2cns}$
 $= -(0101)_2 = -(5)_{10}$

▪ **Example:** $(0, 0100)_{2cns} - (1, 0110)_{2cns} = ?$

o Perform $(0, 0100)_{2cns} + (-(1, 0110)_{2cns})$

o $-(1, 0110)_{2cns} = \text{two's complement of } (1, 0110)_{2cns}$
 $= (0, 1010)_{2cns}$

o $(0, 0100)_{2cns} - (1, 0110)_{2cns} = (0, 0100)_{2cns} + (0, 1010)_{2cns}$
 $= (0, 1110)_{2cns} = +(1110)_2 = +(14)_{10}$

o $+(4)_{10} - (-(10)_{10}) = +(14)_{10}$

Radix Complement Arithmetic (5)

➤ Case 3: $A = -B - C$

- $A = [B]_2 + [C]_2 = 2^n - (B)_2 + 2^n - (C)_2 = 2^n + 2^n - (B + C)_2 = 2^n + [B + C]_2$
- The carry bit (2^n) is discarded.
- An overflow can occur, in which case the sign bit is 0 (*In correct*).

- **Example:** $-(7)_{10} - (8)_{10} = ?$
 - Perform $-(7)_{10} + (-8)_{10}$
 - $-(7)_{10} = -(0111)_2 = (1, 1001)_{2cns}$, $-(8)_{10} = -(1000)_2 = (1, 1000)_{2cns}$
 - $-(7)_{10} - (8)_{10} = (1, 1001)_{2cns} + (1, 1000)_{2cns} = (1, 0001)_{2cns} + \text{carry}$
 $= -(1111)_2 = -(15)_{10}$
- **Example:** $-(12)_{10} - (5)_{10} = ?$
 - Perform $-(12)_{10} + (-5)_{10}$
 - $-(12)_{10} = -(1100)_2 = (1, 0100)_{2cns}$, $-(5)_{10} = -(0101)_2 = (1, 1011)_{2cns}$
 - $-(7)_{10} - (8)_{10} = (1, 0100)_{2cns} + (1, 1011)_{2cns} = (0, 1111)_{2cns} + \text{carry}$
 - **Overflow**, because the sign bit is 0.

Radix Complement Arithmetic (6)

➤ **Example:** $A = (25)_{10}$ and $B = -(46)_{10}$

▪ $A = +(25)_{10} = (0, 0011001)_{2cns}$, $-A = (1, 1100111)_{2cns}$

▪ $B = -(46)_{10} = -(0, 0101110)_2 = (1, 1010010)_{2cns}$, $-B = (0, 0101110)_{2cns}$

▪ $A + B = (0, 0011001)_{2cns} + (1, 1010010)_{2cns} = (1, 1101011)_{2cns} = -(21)_{10}$

▪ $A - B = A + (-B) = (0, 0011001)_{2cns} + (0, 0101110)_{2cns}$
 $= (0, 1000111)_{2cns} = +(71)_{10}$

▪ $B - A = B + (-A) = (1, 1010010)_{2cns} + (1, 1100111)_{2cns}$
 $= (1, 0111001)_{2cns} + \text{carry} = -(0, 1000111)_{2cns}$
 $= -(71)_{10}$

▪ $-A - B = (-A) + (-B) = (1, 1100111)_{2cns} + (0, 0101110)_{2cns}$
 $= (0, 0010101)_{2cns} + \text{carry} = +(21)_{10}$

▪ Note: Carry bit is discarded.

Radix Complement Arithmetic (7)

➤ Summary

Case	Carry	Sign Bit	Condition	Overflow ?
B + C	0	0	$B + C \leq 2^{n-1} - 1$	No
	0	1	$B + C > 2^{n-1} - 1$	Yes
B - C	1	0	$B \leq C$	No
	0	1	$B > C$	No
-B - C	1	1	$-(B + C) \geq -2^{n-1}$	No
	1	0	$-(B + C) < -2^{n-1}$	Yes

➤ When numbers are represented using two's complement number system:

- Addition: Add two numbers.
- Subtraction: Add two's complement of the subtrahend to the minuend.
- Carry bit is discarded, and overflow is detected as shown above.
- Radix complement arithmetic can be used for any radix.

Diminished Radix Complement (1)

- *Diminished radix complement* $[M]_{r-1}$ of a number $(M)_r$ is:

$$[M]_{r-1} = r^n - (M)_r - 1 \quad (1.10)$$

- *One's complement* ($r = 2$):

$$[M]_{2-1} = 2^n - (M)_2 - 1 \quad (1.11)$$

- *Example*: One's complement of $(01100101)_2$

$$\begin{aligned} [M]_{2-1} &= 2^8 - (01100101)_2 - 1 \\ &= (100000000)_2 - (01100101)_2 - (00000001)_2 \\ &= (10011011)_2 - (00000001)_2 \\ &= (10011010)_2 \end{aligned}$$

Diminished Radix Complement (2)

➤ **Example:** Nine's complement of (40960)

$$\begin{aligned}
 [M]_{2-1} &= 10^5 - (40960)_{10} - 1 \\
 &= (100000)_{10} - (40960)_{10} - (00001)_{10} \\
 &= (59040)_{10} - (00001)_{10} \\
 &= (59039)_{10}
 \end{aligned}$$

➤ **Algorithm 1.6** Find $[M]_{r-1}$ given $(M)_r$.

Replace each digit a_i of $(M)_r$ by $r - 1 - a_i$. Note that when $r = 2$, this simplifies

to complementing each individual bit of $(M)_r$.

➤ Radix complement and diminished radix complement of a number (M) :

$$[M]_r = [M]_{r-1} + 1 \quad (1.12)$$

Diminished Radix Complement Arithmetic (1)

- Operands are represented using diminished radix complement number system.
- The carry, if any, is added to the result (*end-around carry*).
- **Example:** Add $+(1001)_2$ and $-(0100)_2$.
One's complement of $+(1001) = 01001$
One's complement of $-(0100) = 11011$
 $01001 + 11011 = 100100$ (carry)
Add the carry to the result: correct result is 00101 .
- **Example:** Add $+(1001)_2$ and $-(1111)_2$.
One's complement of $+(1001) = 01001$
One's complement of $-(1111) = 10000$
 $01001 + 10000 = 11001$ (no carry, so this is the correct result).

Diminished Radix Complement Arithmetic (2)

➤ **Example:** Add $-(1001)_2$ and $-(0011)_2$.

One's complement of the operands are: 10110 and 11100

$10110 + 11100 = 110010$ (carry)

Correct result is $10010 + 1 = 10011$.

➤ **Example:** Add $+(75)_{10}$ and $-(21)_{10}$.

Nine's complements of the operands are: 075 and 978

$075 + 978 = 1053$ (carry)

Correct result is $053 + 1 = 054$

➤ **Example:** Add $+(21)_{10}$ and $-(75)_{10}$.

Nine's complements of the operands are: 021 and 924

$021 + 924 = 945$ (no carry, so this is the correct result).

Computer Codes (1)

- **Code** is a systematic use of a given set of symbols for representing information.
- **Example:** Traffic light (Red: stop, Yellow: caution, Green: go).
- **Numeric Codes**
 - To represent numbers.
 - Fixed-point and floating-point number.
- **Fixed-point Numbers**
 - Used for signed integers or integer fractions.
 - Sign magnitude, two's complement, or one's complement systems are used.
 - Integer: (Sign bit) + (Magnitude) + (Implied radix point)
 - Fraction: (Sign bit) + (Implied radix point) + (Magnitude)

Computer Codes (2)

➤ *Excess or Biased Representation*

- An excess- K representation of a code C : Add K to each code word C .
- Frequently used for the exponents of floating-point numbers.
- Excess-8 representation of 4-bit two's complement code: Table 1.8

Floating Point Numbers (1)

- $N = M * r^E$, where (1.13)
 - M (mantissa or significand) is a significant digits of N
 - E (exponent or characteristic) is an integer exponent.

- In general, $N = \pm (a_{n-1} \dots a_0 . a_{-1} \dots a_{-m})_r$ is represented by
 - $N = \pm (.a_{n-1} \dots a_{-m})_r * r^n$

- M is usually represented in sign magnitude:
 - $M = (S_M . a_{n-1} \dots a_{-m})_{rsm}$, where (1.14)
 - $(.a_{n-1} \dots a_{-m})_r$ represents the magnitude $(-1)^{S_M} \times (.a_{n-1} \dots a_{-m})_r$
 - $S_M =$ (0: positive, 1: negative) (1.15)

Floating Point Numbers (2)

- E is usually coded in excess- K two's complement.
- K is called a bias and usually selected to be 2^{e-1} (e is the number of bits).
- So, biased E is:
 - $-2^{e-1} < E < 2^{e-1}$
 - $0 < E + 2^{e-1} < 2^e$
- Excess- K form of E is written as: $E = (b_{e-1}, b_{e-2} \dots b_0)_{\text{excess-}K}$ (1.16)
where b_{e-1} is the sign bit.
- Combining Eqs. (1.14) and (1.16), we have

$$N = (S_M b_{e-1} b_{e-2} \dots b_0 a_{n-1} \dots a_m)_r \quad (1.17)$$
 representing $N = \quad (1.18)$
- The number 0 is represented by an all-zero word.

Floating Point Numbers (3)

- Multiple representations of a given number:

$$N = M * r^E \quad (1.19)$$

$$= (M / r) * r^{E+1} \quad (1.20)$$

$$= (M * r) * r^{E-1} \quad (1.21)$$

- **Example:** $M = +(1101.0101)_2$

$$M = +(1101.0101)_2$$

$$= (0.11010101)_2 * 2^4 \quad (1.22)$$

$$= (0.011010101)_2 * 2^5 \quad (1.23)$$

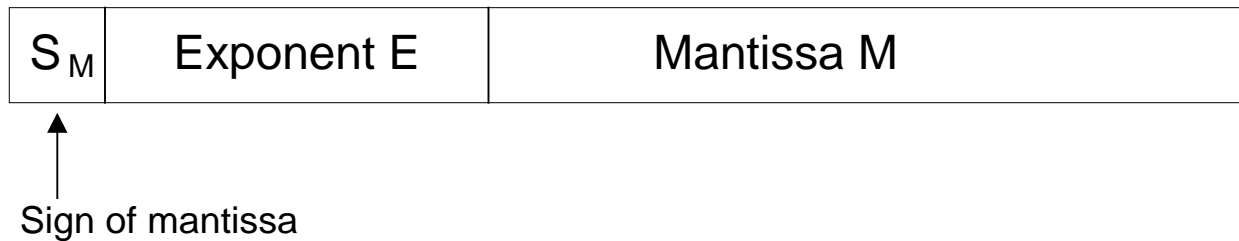
$$= (0.0011010101)_2 * 2^6 \quad (1.24)$$

- **Normalization** is used for a unique representation: mantissa has a nonzero value in its MSD position.
- Eq. 1.22 gives the normalization representation of M .

Floating Point Numbers (4)

➤ *Floating-point Number Formats*

- Typical single-precision format



- Typical extended-precision format



Floating Point Numbers (5)

➤ **Example:** $N = (101101.101)_2$, where $n + m = 10$ and $e = 5$. Assume that a normalized sign magnitude fraction is used for M and that Excess-16 two's complement is used for E .

- $N = (101101.101)_2 = (0.101101101)_2 \cdot 2^6$
- $M = +(0.1011011010)_2 = (0.1011011010)_{2sm}$
- $E = +(6)_{10} = +(0110)_2 = (00110)_{2cns}$
- Add the bias $16 = (10000)_2$ to E
 - $E = 00110 + 10000 = 10110$
 - So, $E = (1, 0110)_{excess-16}$
- Combining M and E , we have
 - $N = (0, 1, 0110, 1011011010)_{fp}$

Characters and Other Codes (1)

➤ To represent information as strings of alphanumeric characters.

➤ *Binary Coded Decimal (BCD)*

- Used to represent the decimal digits 0 - 9.
- 4 bits are used.
- Each bit position has a weight associated with it (*weighted code*).
- Weights are: 8, 4, 2, and 1 from MSB to LSB (called 8421 code).
- BCD Codes:
0: 0000 1: 0001 2: 0010 3: 0011 4: 0100
5: 0101 6: 0110 7: 0111 8: 1000 9: 1001
- Used to encode numbers for output to numerical displays
- Used in processors that perform decimal arithmetic.

▪ *Example:* $(9750)_{10} = (1001011101010000)_{BCD}$

Characters and Other Codes (2)

➤ **ASCII** (American Standard Code for Information Interchange)

- Most widely used character code.
- See Table 1.11 for 7-bit ASCII code.
- The eighth bit is often used for error detection (parity bit)
- **Example:** ASCII code representation of the word *Digital*

<u>Character</u>	<u>Binary Code</u>	<u>Hexadecimal Code</u>
D	1000100	44
i	1101001	69
g	1100111	67
i	1101001	69
t	1110100	74
a	1100001	61
l	1101100	6C

Characters and Other Codes (3)

➤ *Gray Code*

- *Cyclic code*: A circular shifting of a code word produces another code word.
- *Gray code*: A cyclic code with the property that two consecutive code words differ in only 1 bit (the *distance* between the two code words is 1).
- Gray code for decimal numbers 0 - 15: See Table 1.12

Error Detection Codes and Correction codes(1)

- **An error.** An incorrect value in one or more bits.
- **Single error.** An incorrect value in only one bit.
- **Multiple error.** One or more bits are incorrect.
- Errors are introduced by hardware failures, external interference (noise), or other unwanted events.
- **Error detection/correction code:** Information is encoded in such a way that a particular class of errors can be detected and/or corrected.
- Let I and J be n -bit binary information words
 - $w(I)$: the number of 1's in I (*weight*)
 - $d(I, J)$: the number of bit positions in which I and J differ (*distance*)
- **Example:** $I = (01101100)$ and $J = (11000100)$
 - $w(I) = 4$ and $w(J) = 3$
 - $d(I, J) = 3$.

Error Detection Codes and Correction Codes(2)

➤ General Properties

- *Minimum distance, d_{min} , of a code C: for any two code words I and J in C,*

$$d(I, J) > d_{min}$$

- *A code provides t error correction plus detection of s additional errors if and only if the following inequality is satisfied.*

$$2t + s + 1 \leq d_{min} \quad (1.25)$$

- ***Example:***

- *Single-error detection (SED): $s = 1, t = 0, d_{min} = 2.$*

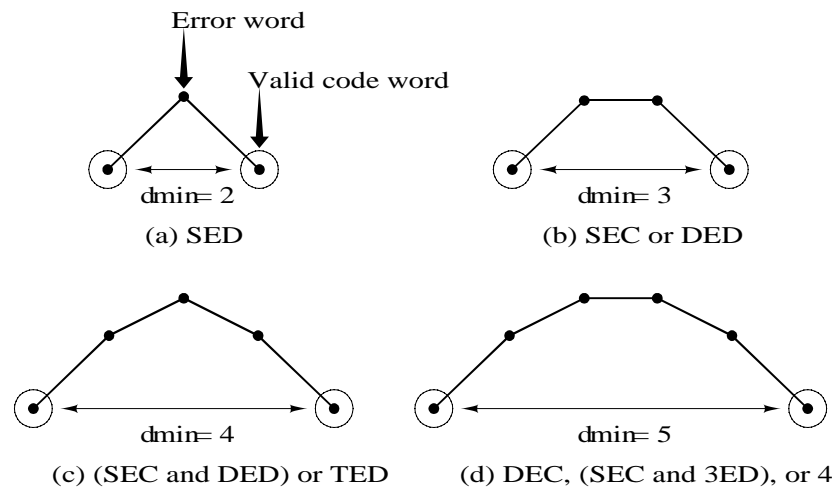
- *Single-error correction (SEC): $s = 0, t = 1, d_{min} = 3.$*

- *Single-error correction and double-error detection (SEC and DED):*

$$s = t = 1, d_{min} = 4.$$

Error Detection Codes and Correction Codes(3)

- Relationship between the minimum distance between code words and the ability to detect and correct errors:

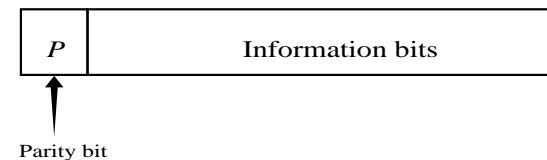


Error Detection Codes and Correction Codes(4)

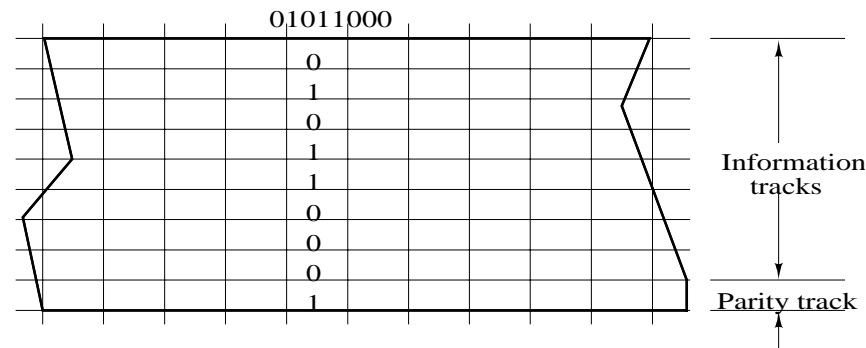
➤ Simple Parity Code

- Concatenate ($|$) a *parity bit*, P , to each code word of C .

- Odd-parity code*: $w(P|C)$ is odd.
- Even-parity code*: $w(P|C)$ is even.



- Parity coding on magnetic tape:



Error Detection Codes and Correction Codes(5)

- *Example:* Odd-parity code for ASCII code characters:

Character	ASCII Code	Odd-parity Code
0	0110000	10110000
X	1011000	01011000
=	0111100	1111100
BEL	0000111	00000111

- Error detection: Check whether a code word has the correct parity.
- Single-error detection code ($d_{\min} = 2$).

➤ *Two-out-of-Five Code*

- Each code word has exactly two 1's and three 0's.
- Detects single errors and multiple errors in adjacent bits.

Hamming Codes (1)

➤ Multiple check bits are employed.

Each check bit is defined over (or covers) a subset of the information bits.

Subsets overlap so that each information bit is in at least two subsets.

d_{min} is equal to the weight of the minimum-weight nonzero code word.

➤ **Hamming Code 1** (Table 1.14)

- $d_{min} = 3$, single error correction code.

- Let \mathcal{C} the set of all code words:

an error word with single error: c_e

the correct code word for the error word: c

then, $d(c_e, c) = 1$ and $d(c_e, w) > 1$ for all other $w \in \mathcal{C}$ (see Table 1.15)

- So, a single error can be detected and corrected by finding out the code word which differs in 1 bit position from the error word.

Hamming Codes (2)

- A code word consists of 4 information bits and 3 check bits:

$$c = (i_3 i_2 i_1 i_0 c_2 c_1 c_0)$$

- Each check bit covers:

$$c_2: i_3, i_2, i_1 \quad c_1: i_3, i_2, i_0 \quad c_0: i_3, i_1, i_0$$

- This relationship is specified by the *generating matrix, G*:

$$G = \begin{bmatrix} 1000111 \\ 0100110 \\ 0010101 \\ 0001011 \end{bmatrix} = \begin{bmatrix} 1000p_{11}p_{12}p_{13} \\ 0100p_{21}p_{22}p_{23} \\ 0010p_{31}p_{32}p_{33} \\ 0001p_{41}p_{42}p_{43} \end{bmatrix} \quad (1.26)$$

- Encoding of an information word i to produce a code word, c :

$$c = iG \quad (1.27)$$

Hamming Codes (3)

- Decoding can be done using the *parity-check matrix, H*:

$$H = \begin{bmatrix} p_{11} & p_{21} & p_{31} & p_{41} & 100 \\ p_{12} & p_{22} & p_{32} & p_{42} & 010 \\ p_{13} & p_{23} & p_{33} & p_{43} & 001 \end{bmatrix} = \begin{bmatrix} 1110100 \\ 1101010 \\ 1011001 \end{bmatrix} \quad (1.28)$$

- H matrix is can be derived from G matrix.
- An n -tuple c is a code word generated by G if and only if

$$Hc^T = 0 \quad (1.29)$$
- Let d be a data word corresponding to a code word c , which has been corrupted by an error pattern e . Then

$$d = c + e \quad (1.30)$$
- Decoding:
 - Compute the syndrome, s , of d using H matrix.
 - s tells the position of the erroneous bit.

Hamming Codes (4)

- Computation of the syndrome:

$$s = Hd^T \quad (1.31)$$

$$= H(c + e)^T$$

$$= Hc^T + He^T$$

$$= 0 + He^T$$

$$= He^T \quad (1.32)$$

Note: All computations are performed using *modulo-2 arithmetic*.

- See Table 1.16 for the syndromes and error patterns.

Hamming Codes (5)

➤ Example : $i=0011$

$$c = iG = [0011] \begin{bmatrix} 1000111 \\ 0100110 \\ 0010101 \\ 0001011 \end{bmatrix} = [0011110]$$

If an error exists in 7th bit ,syndrome computes it.

$$s = Hd^T = \begin{bmatrix} 1110100 \\ 1101010 \\ 1011001 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} = [111]$$

Hamming Codes (6)

➤ *Hamming Code 2* (Table 1.14)

- $d_{min} = 4$, single error correction and double-error detection.
- The generator and parity-check matrices are:

$$(1.34) \quad G = \begin{bmatrix} 10000111 \\ 01001110 \\ 00101101 \\ 00011011 \end{bmatrix} \quad (1.33) H = \begin{bmatrix} 01111000 \\ 11100100 \\ 11010010 \\ 10110001 \end{bmatrix}$$

Odd-weight-column code:

- H matrix has an odd number of ones in each column.
- Example: Hamming Code 2.
- Has many properties; single-error correction, double-error detection, multiple-error detection, low cost encoding and decoding, etc.

Hamming Codes (7)

- Hamming codes are most easily designed by specifying the H matrix.
- For any positive integer $m \geq 3$, there exists an (n, k) SEC Hamming code with the following properties:
 - Code length: $n = 2^m - 1$
 - Number of information bits: $k = 2^m - m - 1$
 - Number of check bits: $n - k = m$
 - Minimum distance: $d_{min} = 3$
- The H matrix is an $n \times m$ matrix with all nonzero m -tuples as its column.
- A possible H matrix for a (15, 11) Hamming code, when $m = 4$:

$$H = \begin{bmatrix} 111101110001000 \\ 111011001100100 \\ 110110100110010 \\ 101110011010001 \end{bmatrix} \quad (1.35)$$

Hamming Codes (8)

➤ **Example:** A Hamming code for encoding five ($k = 5$) information bits.

- Four check bits are required ($m = 4, k = 2^4 - 4 - 1 = 11 > 5$). So, $n = 9$.
- A $(9, 5)$ code can be obtained by deleting six columns from the $(15, 11)$ code shown above.
- The H and G matrices are:

$$H = \begin{bmatrix} 111101000 \\ 111010100 \\ 110110010 \\ 101110001 \end{bmatrix}$$

(1.36)

$$G = \begin{bmatrix} 100001111 \\ 010001110 \\ 001001101 \\ 000101011 \\ 000010111 \end{bmatrix}$$

(1.37)

Summary

- Decimal, Binary, Octal, Hexadecimal number systems.
- Base conversion
- Arithmetic
 - Negative numbers representation
 - Fixed point numbers
 - Floating point numbers
- Codes BCD, Gray, ASCII
- Error detection and Error correction